Project 3 CMSC 630

Image Classification

Overview

This program is designed to classify a set of images of cells into their respective cell type (cyl, inter, let, mod, para, super, or svar). The first set in the chain, images from the dataset are preprocessed and have their features extracted to a csv along with their labels extracted from their filename (e.g. `cyl01.BMP`). In the second part, the csv file of features and labels is read and K nearest neighbors is run on the dataset with k-fold cross validation, outputting the results accuracy of the cross validation. A TOML file in the root directory of the source code is used to configure the operations such as where the outputted csv file should go and what the value of K should be. All important operations are implemented from scratch except for array operations that use the third-party mathematics library `numpy`.

Usage

```
Usage: main.py [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

  CMSC 630 Image Analysis Project Part 3

Options:
  -c, --config PATH  [default: config.toml]
  --help             Show this message and exit.

Commands:
  predict     Use KNN to perdict the label of a new image
  preprocess  Perform feature extraction on initial dataset
  test        Run KNN on dataset to evaluate performance
```

Execution

```
git clone https://github.com/jonaylor89/image-classification.git
cd image-classification
pip3 install --user pipenv
pipenv install
pipenv run python main.py preprocess test
```

Or

```
docker run -it -v $HOME/Repos/CMSC_630_Project_3/datasets:/app/datasets jonaylor/cmsc_project_3
```

(this should pull the image already on dockerhub so the image won't be built locally)

Feature Extraction & KNN

The features extracted were the area of the cluster, the entropy of the image, the mean of the histograms, and the radius of the smallest enclosing circle. The area is calculated by counting the number of pixels belonging to a specific cluster on a morphologically opened image. To find the entropy of an image, the probability of each pixel value is combined to a single scalar value. The mean of the histogram is calculated on a histogram of the occurrences of each pixel value. Finally, the radius of the smallest enclosing circle for a cluster is estimated using the bouncing bubble algorithm on the morphologically opened image. The features are calculated in series for an image, normalized, and concatenated to the label before being saved to the features csv. Testing is performed using k-fold cross validation. The dataset is split into k number of folds (10 by default) and tested by fold. Each fold's training data is feed through KNN to determine its predicted label and tested against the testing portion of the fold. K Nearest Neighbors takes the records in the test set and compares them to the rest of the dataset using euclidean distance. Depending on the value of k, the k closest records in the dataset are taken and the most frequent label on those neighbors is what gets predicted for the test row.

Implementation

The programming language of choice for this project was python. The high level reason for making the decision to write everything in python was that it gives the ability to rapidly develop and integrate each operation as well as for python's `numpy` library which allows for idiomatic and fast array operations. Python's fairly obvious downside is its speed. To mitigate the problem of speed for the image operations, `numba` , a third-party python library used for mathematics, is being used. `numba` has a python function decorator for just-in-time compiling functions to machine code before executing. Using this decorator on functions that use heavy math and looping (i.e. preprocessing) provides major speed increases with speeds similar to using a lower level compiled language like C/C++ or Rust. Compilation time effects the first image for preprocessing, but every image thereafter uses the precompiled machine code. Image batches, rather than being operated on synchronously, are thrown into a process pool where a preconfigured number of worker processes pulls images off a queue and runs them through the preprocessing/feature extraction pipeline.

```
# Preprocessing only (generates csv)
pipenv run python main.py preprocess

# Testing only (Requires the features csv)
pipenv run python main.py test

# Commands can be chained to run both preprocessing and testing
pipenv run python main.py preprocess test

# Predict a label for a new image
pipenv run python main.py predict /path/to/img
```

Full Third-Party Dependency List

```
# Pipefile
[packages]
pillow = "*"     # reading and writing images
numpy = "*"      # fast array operations
click = "*"      # command line interface utility
numba = "*"      # just-in-time compiler for operations
toml = "*"       # reading configuration file
tqdm = "*"       # progress bar
```

These can be found in the Pipfile at the root of the source code

Results

Below is a sample of what the results reported from the cross fold validation for different values of k. The results obtained on the dataset seemed to indicate that the features extracted from the images weren't distinguishable enough to classify the cells as their respective type. Test on more values of k didn't seem to improve the results of the classification. The range for the mean accuracies between k values around 8% to 25%.

```
[INFO] k=1
  Scores: ['4.082%', '12.245%', '6.122%', '16.327%', '10.204%', '10.204%', '6.122%', '8.163%', '10.204%', '8.163%']
  Mean Accuracy: 9.184%


[INFO] k=2
  Scores: ['14.286%', '10.204%', '18.367%', '18.367%', '12.245%', '28.571%', '16.327%', '26.531%', '28.571%', '26.531%']
  Mean Accuracy: 20.000%


[INFO] k=3
  Scores: ['16.327%', '20.408%', '20.408%', '14.286%', '24.490%', '24.490%', '22.449%', '8.163%', '20.408%', '26.531%']
  Mean Accuracy: 19.796%


[INFO] k=4
  Scores: ['10.204%', '24.490%', '14.286%', '22.449%', '16.327%', '16.327%', '28.571%', '30.612%', '12.245%', '18.367%']
  Mean Accuracy: 19.388%


[INFO] k=5
  Scores: ['14.286%', '18.367%', '10.204%', '2.041%', '16.327%', '6.122%', '14.286%', '6.122%', '8.163%', '4.082%']
  Mean Accuracy: 10.000%
```